Identify Fraud from Enron Emails

Introduction:-

Enron was formed in 1985 by Kenneth Lay after the merging of Houston Natural Gas and InterNorth. However, the scandal that was publicised in October 2001 lead to its bankruptcy. A staff of executives lead by Jeffrey Skilling managed to hide the company's bad debts from their shareholders by using accounting loopholes, special purpose entities and poor financial reporting. Enron remains as one of the biggest bankruptcy in American history.

This project will use the machine learning algorithm to predict a 'poi' (person of interest) involved in this scandal. The algorithm used has to achieve a precision and recall score of at least 0.3. I will be running the algorithm in a file called poi_id.py

Dataset Information:-

This dataset was downloaded from: https://www.cs.cmu.edu/~./enron/

Overall, there's 146 datapoints with 21 features in this dataset. The features can be divided into three categories:-

financial features: ['salary', 'deferral_payments', 'total_payments', 'loan_advances', 'bonus', 'restricted_stock_deferred', 'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options', 'other', 'long_term_incentive', 'restricted_stock', 'director_fees'] (all units are in US dollars)

email features: ['to_messages', 'email_address', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi'] (units are generally number of emails messages; notable exception is 'email_address', which is a text string)

POI label: ['poi'] (boolean, represented as integer)

Amongst these 146 samples, 18 of them were identified as 'POI' (per of interests) while 128 is non-POI.

All the features in this dataset has a missing value (denote with 'NaN') except 'poi'. The highest feature with missing value is 'loan_advances'. Next to 'poi, 'total_stock_value' is the lowest feature with the missing value.

Outliers Removal

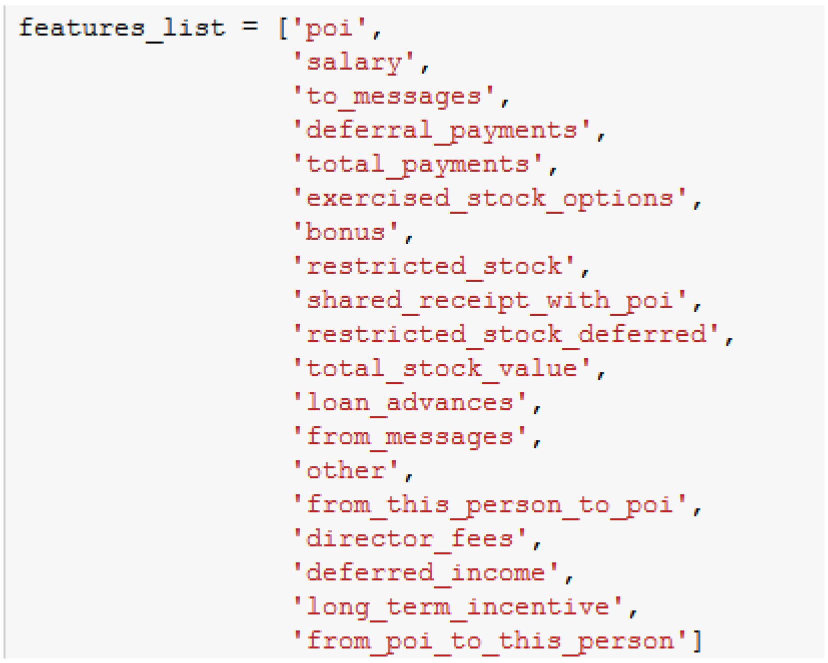
Three outliers were removed from this dataset. They are:-

1. 'TOTAL': This key serves as a value of the financial payments and it does not refer to any particular individual in this dataset.
2. 'THE TRAVEL AGENCY IN THE PARK': This name sounds like a company name and it does not seem to represent any individual.
3. 'LOCKHART EUGENE E': This person has missing value ('NaN') in all of his features.

After removing the outliers, only 143 records remained in this dataset.

Feature Selection

These are the features that were selected at the start of this project. 'email_address' was removed because it interferes with FeatureFormat code as it doesn't have Boolean or numeric values.



```
features_list = ['poi',
                 'salary',
                 'to_messages',
                 'deferral_payments',
                 'total_payments',
                 'exercised_stock_options',
                 'bonus',
                 'restricted_stock',
                 'shared_receipt_with_poi',
                 'restricted_stock_deferred',
                 'total_stock_value',
                 'loan_advances',
                 'from_messages',
                 'other',
                 'from_this_person_to_poi',
                 'director_fees',
                 'deferred_income',
                 'long_term_incentive',
                 'from_poi_to_this_person']
```

On top of the existing features, two new features were created. They are:-

1. 'poi_email_ratio'

   Higher interactions with 'poi' might identify them as a fellow 'poi'. This works on an assumption the staff of executives lead by Jeffrey Skilling are using emails as a form of communication.

2. ‘salary_ratio’

   This feature calculated the ratio between salary and other benefits. A couple of fraud detection points required you to check reimbursement accounts or excessively large payments. There might be some wrongdoings involved if your benefits are higher than your salary (unless the employer write-off your sales incentives as part of the benefits rather than your core salary).

From accountingweb:

14. Employee Expense Accounts

Employees frequently conceal fraud in their individual expense account reimbursements. These reimbursements should be scrutinized for reasonableness and trends, especially in the area of cash transactions on the expense account.

17. Large Payments to Individuals

Excessively large payments to individuals may indicate instances of fraudulent disbursements.

Feature Selection using SelectKBest

Too many features may lead to overfitting. Overfitting will lead the model to perform well in training set, however, it may perform badly in the test set. And it will also took some processing time, as the algorithm will try to fit every single feature, as it deemed each feature to be equally important as the next.

In the beginning, I’ve tried with a few k-values using the GaussianNB with test_train_split algorithm to split the data randomly. These are the precision and recall scores:-

K values in SelectKBest **(k=number of features)**	**Precision and Recall Scores**

K=18	`precision score Gaussian 0.333333333333`
	
	`recall score Gaussian 0.4`

K=10	`precision score Gaussian 0.4`
	
	`recall score Gaussian`

	`0.4`

K=4	`precision score Gaussian 0.0`
	
	`recall score Gaussian 0.0`

The best results might not come from keeping too many values while removing too much values might resulted in less than ideal scores. Hence, the final k-value I've chosen is 10.

SelectKBest calculated all the features with regards to the dependent variable (in this case, the 'poi') and other independent features in the list will not be influencing each other. These are the scores from SelectKBest:-

```
{'salary': 18.289684043404513, 'poi_email_ratio': 15.988502438971512,
'total_payments': 8.7727777300916792, 'bonus': 20.792252047181535,
'total_stock_value': 24.182898678566879, 'shared_receipt_with_poi':
8.589420731682381, 'exercised_stock_options': 24.815079733218194,
'deferred_income': 11.458476579280369, 'restricted_stock':
9.2128106219771002, 'long_term_incentive': 9.9221860131898225}
```

In total, 11 features were selected to the next steps. The top 10 features from SelectKBest and the 'poi' identifier.

Amongst the two new features created, 'poi_email_ratio' was selected due to its high score, however, 'salary_ratio' did not manage to get into the Top 10 cut.

Feature Scaling using MinMaxScaler

Feature scaling is implemented because I do not want one feature to have an overwhelmed influence to the algorithm compared to the next, equally important feature. Some of the algorithm in this project may not need feature scaling (eg; Naïve Bayes), however, in this project, I do not want the financial features (especially those with huge number) to greatly influence the email features. Hence, Feature Scaling was applied.

Algorithm and Parameters Validation

Validation is a process where the trained model is evaluated against the testing data set. Both the training and the testing data came from the same dataset. The biggest chunk of the data goes to the training set, as the algorithm needs more training for every possible scenario. Then, we can use the testing to validate our algorithm performance. We can simply divide the train and test ratio to 70:30. However, if we are not careful, we might be in danger of overfitting our training set.

With this in mind, I will use the train_test_split algorithm to split the data randomly into training set and test set. The train_test_split algorithm will ensure a randomised training and test set. This is to reduce possible bias as both training and test set will be selected randomly.

Parameters tune for train_test_split algorithm are:-

- Test_size=0.3
- Random_state=42
-

Three algorithm classifiers were selected for this project:-

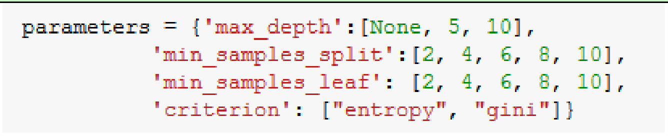
1. Naïve Bayes (GaussianNB from sklearn)
2. Decision Tree (DecisionTreeClassifier from sklearn)
3. Support Vector Machine (SVC from sklearn)

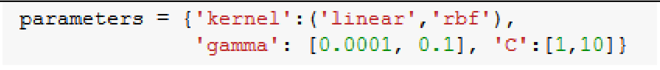
Parameters tuning is important to ensure we get the best possible results from the learning task, as the parameters set at optimal value will enable the algorithm to work in a best possible condition. Some algorithm comes with parameters, and most of them have default values. However, with only using the default values, we might have missed a better combination of the parameters out there. Also, badly tuned parameters could lead us to a false result, where the algorithm performs well while we're tuning them, but it could not replicate the same result when we run the cross-validation test.

According to this site, GaussianNB does not need parameters tuning. However, GridSearchCV algorithm was used to find the best parameters for both Decision Tree and Support Vector Machine algorithm.

Below are the best possible parameters combinations for Decision Tree and SVC.

Algorithm	**Parameters Grid**	**Best Possible Combination**

Decision Tree	 `parameters = {'max_depth':[None, 5, 10],`
	`'min_samples_split':[2, 4, 6, 8, 10],`
	`'min_samples_leaf': [2, 4, 6, 8, 10],`
	`'criterion': ["entropy", "gini"]}`
		`{'min_samples_split' : 8, 'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 2}`

Support Vector Machine	 `parameters = {'kernel':('linear','rbf'),` `'gamma': [0.0001, 0.1], 'C':[1,10]}`	`{'kernel': 'rbf', 'C': 1, 'gamma': 0.0001}`

Evaluation

Two evaluation scores are being used in this project. They are precision score and recall score.

Precision Score: How many selected items are relevant compared to the rest of the items?

Recall Score: How many of those relevant items that were actually selected?

These are the scores when I ran the above classifiers in poi_id.py:-

Classifier	**Precision Score**	**Recall Score**
GaussianNB	0.4	0.4
Decision Tree	0.75	0.6
Decision Tree with parameters tuned	1.0	0.8
SVC	0.0	0.0
SVC with parameters tuned	1.0	0.4

The best score comes from Decision Tree with parameters tuned at 1.0 precision score and 0.8 recall score.

Testing the algorithm with tester.py

Unfortunately, this is not the best score when I ran it with tester.py:-

Classifier	**Results**

GaussianNB	`Accuracy: 0.83613`
	`Precision: 0.36639`
	`Recall: 0.31400`
	`F1: 0.33818`
	`F2: 0.32324`
	`Total predictions: 15000`
	`True positives:  628`
	`False positives: 1086`
	`False negatives: 1372`
	`True negatives: 11914`

Decision Tree with parameters tuned	`Accuracy: 0.80593`
	`Precision: 0.24931`
	`Recall: 0.22650`
	`F1: 0.23736`
	`F2: 0.23072`
	`Total predictions: 15000`
	`True positives:  453`
	`False positives: 1364`
	`False negatives: 1547`
	`True negatives: 11636`

SVC with parameters tuned	Aborted

Both the Decision Tree with parameters tuned and SVC with parameters tuned classifiers were used in the tester.py, as they have a better scores compared to their non-parameters tuned condition. However, their precision and recall scores are lower. Also, the SVC with parameters tuned is running slower than usual (>3600seconds) when you tested it in tester.py, hence, it was aborted due to the bad performance.

Since the results are different, I went back to check the difference between my code and the tester code. The main difference is the tester.py code is using StratisfiedShuffleSplit for their validation. With this in mind, I went back to my code and introduced this same algorithm for my validation.

StratisfiedShuffleSplit was selected due to the small sample size. In this case, it will work better than train_test_split code. Parameters tuned for StratisfiedShuffleSplit is per below:

- folds=1000
- random_state=42

These parameters are consistent with the parameters used in GridSearchCV and the tester.py code.

Below are the results with StratisfiedShuffleSplit:-

Classifier using StratisfiedShuffleSplit	**Precision Score**	**Recall Score**
GaussianNB	0.667	1.0
Decision Tree	0.5	0.5
Decision Tree with parameters tuned	1.0	0.5
SVC	0.0	0.0
SVC with parameters tuned	0.0	0.0

Both the Naïve Bayes and Decision Tree algorithm have the same precision and recall score. Ironically, Decision Tree with parameters tuned did not do well after I used StratisfiedShuffleSplit. Support Vector Machine has the lowest score with this algorithm, regardless of its parameters.

Now it's time to use the tester.py to see how good it will go. Below are the results:-

Classifier	**Results with tester.py**

GaussianNB	`Accuracy: 0.83613`
	`Precision: 0.36639`
	`Recall: 0.31400`
	`F1: 0.33818`
	`F2: 0.32324`
	`Total predictions: 15000`
	`True positives:  628`
	`False positives: 1086`
	`False negatives: 1372`
	`True negatives: 11914`

Decision Tree with parameters tuned	`DecisionTreeClassifier(class_weight=None,`
	`criterion='entropy', max_depth=5,`
	`            max_features=None, max_leaf_nodes=None,`
	`min_samples_leaf=2,`
	`            min_samples_split=2,`
	`min_weight_fraction_leaf=0.0,`
	`            presort=False, random_state=None,`
	`splitter='best')`
	
	`Accuracy: 0.81840`
	`Precision: 0.26972`
	`Recall: 0.21200`
	`F1: 0.23740`
	`F2: 0.22148`
	`Total predictions: 15000`
	`True positives:  424`

	`False positives: 1148`
	`False negatives: 1576`
	`True negatives: 11852`

GaussianNB classifier works better with StratisfiedShuffleSplit algorithm compares with Decision Tree with parameters tuned classifier. The Decision Tree results with tester.py performed worse than the one with poi_id.py. I suspect it may due to the FeatureFormat code. In my code, I applied them before I called the StratisfiedShuffleSplit function. But in the tester.py code, the FeatureFormat was applied inside the StratisfiedShuffleSplit function. That's the difference I am able to see.

Nevertheless, I will select the GaussianNB classifier as my final classifier since it performed consistently well with both StratisfiedShuffleSplit and train_test_split algorithm.

Final Analysis

GaussianNB was selected as the final algorithm since it performed the best with tester.py code. The precision score is 0.36639 and the recall score is 0.314.

Precision in this case is referring to the proportion of the persons who are indeed poi over the number of persons who classified as poi.

Recall in this project is referring to the proportion of persons who are indeed poi and was correctly picked by the classifier over the number of persons who are indeed poi.

Algorithm used for GaussianNB as final classifier:-

Stages	**Algorithm Used**
Feature Selection	SelectKBest from sklearn
Feature Scaling	MinMaxScaler from sklearn
Validation	StratisfiedShuffleSplit from sklearn
Classifier	GaussianNB from sklearn
Evaluator	• Precision_score from sklearn • Recall_score from sklearn

Reference

Enron scandal

https://en.wikipedia.org/wiki/Enron_scandal

http://www.businessinsider.com/largest-bankruptcies-in-american-history-2011-11/?IR=T#enron-6

Dataset download

https://www.cs.cmu.edu/~./enron/

Fraud Detection in Accounting

https://www.accountingweb.com/aa/law-and-enforcement/20-ways-you-can-detect-fraud

sklearn documentation

http://scikit-learn.org/stable/index.html

Parameters tuning for GaussianNB

https://stackoverflow.com/questions/39828535/how-to-tune-guassiannb

Precision and Recall Score in Machine Learning

https://en.wikipedia.org/wiki/Precision_and_recall

Summary of Machine Learning algorithm

https://i.stack.imgur.com/OKOsB.png

Udacity Project Page

https://classroom.udacity.com/nanodegrees/nd002/parts/0021345409/modules/317428862475461
/lessons/3174288624239847/concepts/31803986370923